EPM-RT-2001-02

A Pattern Reordering Approach Based on ' Ambiguity Detection for On-Line Category ' Learning

Éric Granger*
Yvon Savaria*
Pierre Lavoie*

Département de génie électrique

Octobre 2002

ÉCOLE POLYTECHNIQUE MONTRÉAL
Le génie sans frontières
AFFILIÉE À L'UNIVERSITÉ DE MONTRÉAL

EPM/RT-01/02

A Pattem Reordering Approach Based on Ambiguity Detection for On-Line Category Learning

Éric Granger
Yvon Savaria
Pierre Lavoie

Département de génie électrique

Octobre 2001

02001-09
Éric Granger, Yvon Savaria, Pierre Lavoie

Dépôt légal :
Bibliothèque nationale du Québec, 2001
Bibliothèque nationale du Canada, 200 1

EPM-RT-O 1/02
A Pattern Reordering Approach Based on Ambiguity Detection for On-Line Category Learning
par : Éric Granger, Yvon Savaria et Pierre Lavoie
Département de génie électrique
Ecole Polytechnique de Montréal

A Pattern Reordering Approach Based on Ambiguity Detection for On-Line Category Learning

Eric Granger[1,2,3], **Yvon Savaria**[2] **and Pierre Lavoie**[1]

[1] Defence Research Establishment Ottawa
Dept. of National Defence
Ottawa, Ontario, Canada

[2] Dept. of Electrical and Computer Eng.
École Polytechnique de Montréal
Montreal, Quebec, Canada

Index Terms: ambiguity, on-line category learning, partitional clustering, pattern recognition, reject option.

Abstract

Pattern reordering is proposed as an alternative to sequential and batch processing for on-line category learning. Upon detecting that the categorization of a new input is ambiguous, the input is postponed for a predefine time, after which it is reexamined and categorized for good. This approach is shown to improve the categorization performance over purely sequential processing, while yielding a shorter response time, or latency than batch processing. The latency of a typical implementation is derived and compared to lower bounds. Gaussian and softmax models are derived from reject option theory, and are considered for detecting ambiguity and triggering pattern postponement. The latency and Rand Adjusted score of reordered, sequential and batch processing are compared through computer simulation with Gaussian and Iris data sets.

[3]Corresponding author: Defence Research Establishment Ottawa, 3701 Carling Ave., Ottawa, Ontario, K1A 0Z4, Canada, email: Eric.Granger@dreo.dnd.ca, phone: 1-613-991-4513, fax: 1-613-990-8401.

1 Introduction

A number of pattern recognition applications involve high throughput category learning from continuous streams of input patterns. In pattern recognition literature, partitional clustering techniques [2] [11], such as on-line versions of the *k*-means [34] and leader [25] algorithms, are proposed for on-line category learning. Other algorithms include on-line versions of adaptive vector quantization (AVQ) [18] [20] (*e.g.,* generalized Lloyd [33] and Linde-Buzo-Gray (LBG) [31] algorithms) in communication theory, and unsupervised competitive learning (UCL) [21] [22] [23] [30] (*e.g.,* standard UCL networks [1] [10] [35] and Adaptive Resonance Theory (ART) networks [5]) in neural network theory. These algorithms learn input patterns autonomously on the fl , some without prior knowledge of the number and characteristics of categories. The above algorithms do not require long-term storage of input patterns, and lend themselves well to high throughput, parallel implementation.

A common trait of these algorithms is their *sequential processing* of input patterns. When an input pattern is presented, it is compared to the prototype of every category. The category with the best matching prototype (according to some similarity measure) is assigned to the input, and the prototype of this category adapts to novel characteristics of the input. Over a succession of inputs, category prototypes become tuned to different parts of the input space, and implicitly defin inter-category decision boundaries. Therefore, the unknown probability density function of the data source is progressively estimated using a finit number of prototypes.

With sequential processing, a fina decision is taken upon presentation of every input. This implies that, for each decision, only information from prior input patterns is available. In addition, the fact that prototypes are stored rather than prior input patterns implies that learning — the

consequence of each decision — is irreversible. Information is therefore lost in the process.

Limitations of sequential processing are obvious, particularly if the structure of input data clusters tends to be scattered and/or overlapping. For instance, if upon its presentation, an input pattern lies in proximity to a decision boundary separating two or more categories, the clusterer is forced to make a fina decision, despite the *ambiguity*. Irreversible learning for such decisions gives rise to different category structures, and diverging decision boundaries that vary significantl according to the pattern presentation order. Thus, clustering quality depends largely on the context.

The quality of results is generally improved with *batch processing*. It consists of collecting and storing one batch of patterns from the input stream, while the clusterer converges iteratively, over several epochs, on a previously-accumulated batch of patterns. An epoch is one complete presentation of the fi ed-size batch of data, one pattern at a time, to the clusterer. Convergence is attained when the prototypes remain constant for two successive epochs. Batch processing can diminish or eliminate sensitivity to the input presentation order by allowing the clusterer, for example, to recover from some early miss-assignments caused by exposure to ambiguity. Although this approach can yield better results, it entails delays caused by the accumulation of patterns to form batches, and requires data buffering to account for the variability of the computational effort.

In this paper, an alternative called *reordered processing* is proposed for on-line category learning. As a clusterer assigns categories to input patterns, it is granted the ability to postpone, or delay, the categorization (category assignment and learning) when it detects an ambiguity. Delayed patterns are queued, and their categorization is deferred for some fi ed time. The overall effect is a modificatio to the order in which inputs are categorized. Aside from allowing to control the maximum response time, pattern postponement offers the opportunity to circumvent learning for

ambiguous decisions.

The rest of this paper is organized as follows. In Section 2, lower bounds on the latency required for on-line category learning are developed for sequential, batch and reordered processing. Then, practical issues are discussed, and high-level architectures are proposed to implement batch and reordered processing. Elements of reject option theory are developed for the detection of ambiguous category assignment in Section 4. The experimental methodology, and proof-of-concept computer simulations using three neural networks are presented and discussed in Section 5. It is shown that the proportion of patterns that are declared ambiguous is a good indicator of poor clustering quality; and that modifying the order in which input patterns are processed, based on ambiguity detection, can enhance clustering quality for a moderate increase in latency. Finally, reordered processing offers an interesting alternative to batch processing in terms of trading off clustering quality versus response time.

2 Latency in on-line category learning

On-line category learning of a continuous stream of input patterns is performed by a clusterer (*e.g.,* an on-line *k*-means algorithm) which is exploited through one of several data processing schemes. Given our focus on high throughput applications, the input response time incurred by adopting a data processing scheme is a critical consideration. Let input patterns to be categorized arrive one at a time from some source that provides them at a rate that need not be constant. The *latency L* of the category learning is define as the number of input patterns subsequent to an input **a** that are necessary before a fina decision (category) $\mathbf{y}(\mathbf{a})$ can be assigned to **a**. By neglecting the response time of the clusterer, lower bounds on the latency required with a data processing scheme can be

characterized. Lower bounds on the minimum, maximum and average latency required for three data processing schemes — sequential, batch and reordered — are now examined.

A basic approach to learning a continuous data stream is through *sequential processing*. Upon observation of each input pattern **a**, a category can be assigned to **a** without waiting for subsequent patterns.

Batch processing consists in categorizing the input patterns $\{\mathbf{a}\}$ in fi ed-size batches of k patterns. Incoming patterns are accumulated until a group of k successive patterns has been formed. The following patterns are buffered while the previous k patterns are being learned. Batches of incoming data are learned iteratively, over several epochs, until convergence is attained. The category assigned to each pattern **a** in a batch is produced once convergence has been detected, *i.e.,* upon the last one of these epochs.

Reordered processing consists in postponing the learning of input patterns **a** for which category assignment is deemed ambiguous. Each postponed pattern is queued, and categorized following a fi ed latency of d patterns. To bound the latency of reordered patterns, a given pattern can only be postponed once.

The minimum latency $L_{\min}$ for all three data processing schemes must satisfy:

$$L_{\min} \geq 0 \ . \tag{1}$$

This lower bound corresponds to **a** being the last pattern in a batch with batch processing, or **a** not

being postponed with reordered processing. The maximum latency $L_{\max}$ must satisfy:

$$L_{\max} \geq \begin{cases} 0\ ; & \text{sequential processing,} \\ k-1\ ; & \text{batch processing,} \\ d\ ; & \text{reordered processing,} \end{cases} \tag{2}$$

where k is the number of patterns per batch, and d is the user-define queue size. This lower bound corresponds to **a** being the firs pattern in a batch with batch processing, or **a** being postponed with reordered processing. Lastly, it is straightforward to show that the average latency $\overline{L}$ is:

$$\overline{L} \geq \begin{cases} 0\ ; & \text{sequential processing,} \\ \frac{1}{k}\sum_{i=1}^{k}(k-i)\ ; & \text{batch processing,} \\ p_r(\mathbf{a})\cdot d\ ; & \text{reordered processing,} \end{cases} \tag{3}$$

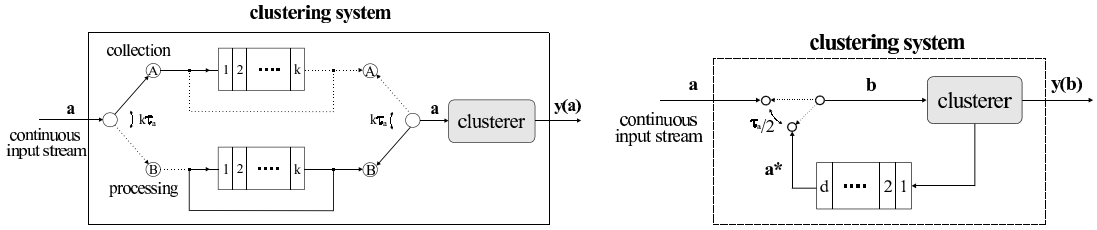
where $p_r(\mathbf{a})$ is the pattern rejection, or postponement rate. This rate depends on the data structure and the rejection criterion.

Reordered processing constitutes a compromise between sequential processing and batch processing. Notice that if $d = (k-1)$ and $p_r(\mathbf{a}) = 50\%$, then the lower bounds on $L_{\max}$ and $\overline{L}$ are equal for batch and reordered processing. If, on the other hand, $d = 0$, then reordered processing reduces to sequential processing.

3 Practical clustering systems

In order to characterize the latency of clustering systems for on-line category learning, assume from here on that they accept input stream patterns at a *regular* interval τ_a. By making abstraction of the

clusterer embedded within this system, the reported latency characterizes the overhead required with a data processing scheme. The rest of this section raises practical issues involved with the implementation of clustering systems, and describes high level architectures that can accommodate batch and reordered processing.



(a) batch processing architecture. (b) reordered processing architecture.

Figure 1: High level architectures of clustering systems for on-line clustering.

3.1 An architecture for batch processing

Batch processing involves incremental convergence over several *batch epochs* δ — the number of successive presentations of a batch of k patterns to the clusterer for convergence. The number of batch epochs required to attain the state of convergence varies from one batch to the next, according to the queue length k, as well as the structure of the specifi data patterns in the batch. The computational effort is therefore variable and somewhat unpredictable. The requirement for convergence is also costly, because at least one whole batch epoch of overhead is required to detect the state of convergence, thus $\delta \geq 2$. Finally, accumulation of incoming data occurs until the clusterer has converged for the current batch of patterns.

One possible architecture to implement batch processing is shown in Figure 1(a). It consists of

two identical fi ed-length queues of k registers that operate concurrently, and a clusterer that can process each pattern within a fi ed time τ_c. Using two parallel queues eliminates the clusterer's delay for accessing a new batch of data. Each queue alternates between a collection (A) and a processing (B) mode. While one of the queues temporarily buffers k incoming patterns from the input stream (A), the other queue stores a batch of k patterns being learned by the clusterer (B).

To circumvent the delay required for detection of convergence, the number of batch epochs, δ, can be bound to a constant value across batches, $\delta \geq 2$. A fina decision $\mathbf{y}(\mathbf{a})$ can be produced immediately after processing of **a**, as part of the last epoch. The processing rate must satisfy:

$$\tau_c \leq \frac{\tau_a}{\delta} \ , \tag{4}$$

which imposes a processing rate constraint on the clusterer that is proportional to the number of batch epochs δ, and prevents overfl ws of the queue operating in collection mode.

Immediately after a queue is switched to the collection mode (A), a pattern **a** from the input stream is stored every τ_a seconds inside its registers. After $k\tau_a$ seconds, the registers are full, and the queue is switched into processing mode (B). In this mode, patterns are shifted right through the clusterer every τ_c seconds. A feedback loop allows repeated presentations. After δ epochs, the registers are reset prior to switching back to collection mode. The latency of the batch processing architecture of Figure 1(a) is the sum of the delays incurred in both modes. In the best case, when **a** is the last of a batch,

$$L_{\min} = k \ , \tag{5}$$

and in the worst case, when **a** is the firs of a batch,

$$L_{\max} = (k-1) + \left\lceil \frac{(\delta-1)k+1}{\delta} \right\rceil \ , \tag{6}$$

The average latency of this architecture is:

$$\overline{L} = \frac{1}{k}\sum_{i=1}^{k}\left\{(k-i)+\left\lceil\frac{(\delta-1)k+i}{\delta}\right\rceil\right\} \quad , \tag{7}$$

It is easily verifie that (5), (6) and (7) satisfy but do not meet the lower bounds (1), (2) and (3).

3.2 An architecture for reordered processing

With a clustering system that uses reordered processing, a pattern having been postponed by d patterns has priority over a pattern from the input stream. This ensures a fi ed worst-case response time of d patterns, but leads to the accumulation of up to d incoming patterns at the clustering system's input. Also, a clusterer embedded within this system requires additional circuitry to detect ambiguity.

One possible architecture to implement reordered processing is shown in Figure 1(b). It consists of a fi ed-length queue composed of d registers, and a clusterer capable of detecting ambiguous category assignments. If an input pattern $\mathbf{a}$ is deemed ambiguous, it is diverted towards the queue and labeled $\mathbf{a}^*$. Shifting this pattern through the queue is equivalent to a fi ed delay that changes the presentation order.

Each pattern $\mathbf{b}$ that is processed by the clusterer is either a new input pattern, $\mathbf{b} = \mathbf{a}$, or a previously-rejected and therefore delayed pattern, $\mathbf{b} = \mathbf{a}^*$. A simple way to schedule processing is to assume that the clusterer's processing time is partitioned into two equal time slices: one slice reserved to process input patterns $\mathbf{a}$, and the other slice reserved to process previously-rejected patterns $\mathbf{a}^*$. To allow alternating between the two sources, the augmented clusterer must be able to process a pattern $\mathbf{b}$ within a fi ed time of τ_c that satisfies

$$\tau_c \leq \frac{\tau_a}{2} \quad . \tag{8}$$

This speed constraint is similar to that of batch processing with $\delta = 2$. With proper synchronization, the need for an input buffer is avoided. With this architecture, the lower bounds (1), (2) and (3) are met.

4 Ambiguity in competitive learning assignments

With sequential processing, each input pattern **a** that is presented to the clusterer is compared to the prototype $\mathbf{w}_j$ representing category j, for $j = 1, 2, ..., N$. The category $j = J$ with the best matching prototype is assigned to input **a**. Two forms of uncertainty can influenc this assignment.

Intra-category uncertainty accounts for the degree of mismatch between **a** and prototype $\mathbf{w}_J$ of category J. For clusterers in which the number of categories N is fi ed (*e.g.,* on-line k-means algorithm), this uncertainty can be reduced at the outset by using greater N values. By contrast, other clusterers (*e.g.,* leader algorithm) regulate the maximum tolerable amount of intra-category uncertainty by imposing a minimum degree of match per assignment. If assignment of category J to **a** does not satisfy the matching threshold, then a new category is initialized to represent **a**.

Inter-category uncertainty or *ambiguity* accounts for the degree of match between **a** and the prototypes of all N categories. Ambiguity grows as **a** lies closer to a decision boundary between category J and any other category $j = 1, 2, ..., N$, $j \neq J$. The symptom of this is multiple prototypes $\mathbf{w}_j$ having an almost equal degree of match with **a**.

With reordered processing, ambiguity is employed as a criterion for postponing pattern categorization. The clusterer is equipped with the ability to detect ambiguity during category assignment (*i.e.,* before prototype adjustment). The rest of this section concerns the measurement, from a statistical standpoint, of ambiguity. Given the focus on high throughput clustering, particular em-

phasis is placed on measures with low computational complexity.

4.1 Bayes decision theory and the reject option

In statistical pattern recognition, the *reject option* [7] [8] [16] provides a framework for detecting the ambiguous assignment of a class to an input pattern. By exercising an option to reject, a recognition system can abstain from making a meaningless classification and submit the rejected input pattern for exceptional handling such as manual inspection.

If the underlying statistics of the pattern recognition problem are fully known, the Bayes decision procedure assigns 1-out-of-N possible classes to input pattern $\mathbf{a}$ using the maximum *a posteriori* probability decision rule:

$$J = \arg\max\{p(j \mid \mathbf{a}) : j = 1,2,...,N\}\,, \tag{9}$$

where $0 \leq p(j \mid \mathbf{a}) \leq 1$ and $\sum_{j=1}^{N} p(j \mid \mathbf{a}) = 1$. The *a posteriori* probability $p(j \mid \mathbf{a})$ that class j generated input $\mathbf{a}$ is computed according to the Bayes theorem:

$$p(j \mid \mathbf{a}) = \frac{P_j\, p(\mathbf{a} \mid j)}{p(\mathbf{a})} = \frac{P_j\, p(\mathbf{a} \mid j)}{\sum_{k=1}^{N} P_k\, p(\mathbf{a} \mid k)}\;;\; j = 1,2,...,N\,, \tag{10}$$

where P_j is the *a priori* probability of class j, with $0 \leq P_j \leq 1$ and $\sum_{j=1}^{N} P_j = 1$, $p(\mathbf{a} \mid j)$ is the conditional probability density function (p.d.f.) of the input $\mathbf{a}$ given class j, and $p(\mathbf{a}) = \sum_{j=1}^{N} P_j\, p(\mathbf{a} \mid j)$ is the unconditional p.d.f. of input $\mathbf{a}$ (mixture density function). Eq. (9) define decision boundaries that yield the minimum probability of miss-classificatio [3] [13] [17].

The degree of ambiguity regarding the decision rule of Eq. (9) can be measured in terms of the *conditional error* given input $\mathbf{a}$ [7] [8] [12] [16]:

$$r_J(\mathbf{a}) \;=\; 1 - \max\{p(j \mid \mathbf{a}) : j = 1,2,...,N\}$$

$$
\begin{aligned}
&= 1 - p(J \mid \mathbf{a}) \\
&= \sum_{j=1, j \neq J}^{N} p(j \mid \mathbf{a}).
\end{aligned} \tag{11}
$$

Assignment of class J to input $\mathbf{a}$ is define to be ambiguous if $r_J(\mathbf{a})$ is greater than or equal to a rejection threshold $\gamma \in [0, 1]$:

$$
r_J(\mathbf{a}) \geq \gamma \,. \tag{12}
$$

From Eq. (10):

$$
\begin{aligned}
\frac{\sum_{j=1, j \neq J}^{N} P_j \, p(\mathbf{a} \mid j)}{\sum_{j=1}^{N} P_j \, p(\mathbf{a} \mid j)} &\geq \gamma \\
\left(\frac{1 - \gamma}{\gamma} \right) &\geq \frac{P_J \, p(\mathbf{a} \mid J)}{\sum_{j=1, j \neq J}^{N} P_j \, p(\mathbf{a} \mid j)} \,.
\end{aligned} \tag{13}
$$

Since the conditional error given $\mathbf{a}$ ranges from $r_J(\mathbf{a}) = 0$ (the assignment is completely unambiguous, with $p(J \mid \mathbf{a}) = 1$) to $r_J(\mathbf{a}) = \frac{N-1}{N}$ (the assignment is completely ambiguous or random, with $p(J \mid \mathbf{a}) = \frac{1}{N}$), the rejection threshold must satisfy $\gamma \in (0, \frac{N-1}{N}]$ to be consistent with Eq. (13).

Eq. (13) define a region of ambiguity, $L_r(\gamma) = \{\mathbf{a} : r_J(\mathbf{a}) \geq \gamma\}$, of the input space, that is, where class assignments are considered to be ambiguous. The size of this region grows as γ is decreased. For a given γ, the size of the region also depends on the shape of $r_J(\mathbf{a})$ at the decision boundary of class J, and hence on the class distribution in the input space. The rate of rejection is:

$$
p_r(\mathbf{a}) = \int_{L_r(\gamma)} p(\mathbf{a}) \, \mathrm{d}\mathbf{a} \tag{14}
$$

The option to reject patterns that fall in $L_r(\gamma)$ provides a means of reducing the number of classificatio errors. Patterns that would have been correctly classifie may also be rejected. The performance level of a pattern recognition system with reject option is therefore characterized by the trade-off between error and rejection rates [8] [16] [24].

4.2 Extensions for on-line category learning

In practical applications, the probabilities required to implement the reject option are usually unknown. They must be estimated based on a clusterer's response to input patterns. A clusterer may be subjected to different environments, with more or less prior information on the underlying data structure. Two environments are now considered. In the first clusters of data are obtained from sources having identically distributed Gaussian distributions, where all variables are independent and have equal variance σ^2. In the second, no prior assumption is made about the underlying input data structure, and data clusters are modeled implicitly. Eq. (13) is now developed for each one of these environments.

4.2.1 Gaussian rejection model

The result of on-line category learning can be interpreted as a model of the statistical distribution of the input data. Assume that the probability distribution of the data $p(\mathbf{a})$ can be modeled as a mixture of conditional density functions $p(\mathbf{a} \mid j)$ for $j = 1, 2, ..., N$: $p(\mathbf{a}) = \sum_{j=1}^{N} P_j \, p(\mathbf{a} \mid j)$ [3] [13]. Each component $p(\mathbf{a} \mid j)$ corresponds to the probability of input $\mathbf{a}$, given category j, whereas mixing parameter P_j corresponds to the a priori probability of category j.

Assuming that the noise is Gaussian, the conditional density functions $p(\mathbf{a} \mid j)$ are define by independent normal distribution functions. In an M-dimensional space, the multivariate normal probability distribution is:

$$p(\mathbf{a} \mid j) = \frac{1}{(2\pi)^{\frac{M}{2}} |\Lambda_j|^{\frac{1}{2}}} \exp\left\{ -\frac{1}{2} (\mathbf{a} - \mu_j)^T \Lambda_j^{-1} (\mathbf{a} - \mu_j) \right\}, \tag{15}$$

where μ_j and Λ_j are the mean vector and covariance matrix for category j. Therefore each category j is represented as an M-dimensional ellipsoid centered at μ_j.

If the data clusters are generated by sources with the same Gaussian noise, and all categories have equal covariance matrices ($\Lambda_j = \Lambda$ for $j = 1,2,...,N$), then the decision boundaries are hyperplanar. If all variables are statistically independent and have equal variance σ^2, then Λ is a diagonal matrix ($\Lambda = \sigma^2\mathbf{I}$, where $\mathbf{I}$ is the identity matrix, and $|\Lambda| = \prod_{i=1}^{M}\sigma_i^2 = \sigma^{2M}$), and distributions are hyperspherical. Lastly, if all categories have equal *a priori* probabilities ($P_j = 1/N$ for $j = 1,2,...,N$), then the maximum *a posteriori* probability decision rule given in Eq. (9) reduces to template matching:

$$\begin{aligned} J &= \arg\max\left\{\frac{1}{N(2\pi\sigma^2)^{\frac{M}{2}}}\exp\left\{-\frac{1}{2}(\mathbf{a}-\mu_j)^T\Lambda_j^{-1}(\mathbf{a}-\mu_j)\right\} : j = 1,2,...,N\right\} \\ &= \arg\max\left\{-\frac{1}{2}\sum_{i=1}^{M}\left(\frac{a_i-\mu_{ji}}{\sigma_{ji}}\right)^2 : j = 1,2,...,N\right\} \\ &= \arg\min\{||\mathbf{a}-\mu_j||^2 : j = 1,2,...,N\}\,, \end{aligned} \tag{16}$$

where $||\mathbf{a}-\mu_j||$ is the Euclidean distance. Data clusters are therefore modeled explicitly as hyperspherical normal distributions, which constitute estimates of the conditional probabilities $\{p(\mathbf{a} \mid j) : j = 1,2,...,N\}$.

In light of the previous assumptions, the reject option of Eq. (13) is reduced to:

$$\begin{aligned} \left(\frac{1-\gamma}{\gamma}\right) &\geq \frac{\frac{1}{N(2\pi\sigma^2)^{\frac{M}{2}}}\exp\{-\frac{1}{2}(\mathbf{a}-\mu_J)^T\Lambda_J^{-1}(\mathbf{a}-\mu_J)\}}{\sum_{j=1,j\neq J}^{N}\frac{1}{N(2\pi\sigma^2)^{\frac{M}{2}}}\exp\{-\frac{1}{2}(\mathbf{a}-\mu_j)^T\Lambda_j^{-1}(\mathbf{a}-\mu_j)\}} \\ &= \frac{\exp\left\{-\frac{1}{2}\sum_{i=1}^{M}\left(\frac{a_i-\mu_{Ji}}{\sigma_{Ji}}\right)^2\right\}}{\sum_{j=1,j\neq J}^{N}\exp\left\{-\frac{1}{2}\sum_{i=1}^{M}\left(\frac{a_i-\mu_{ji}}{\sigma_{ji}}\right)^2\right\}} \\ &= \frac{\exp\left\{-\frac{||\mathbf{a}-\mu_J||^2}{2\sigma^2}\right\}}{\sum_{j=1,j\neq J}^{N}\exp\left\{-\frac{||\mathbf{a}-\mu_j||^2}{2\sigma^2}\right\}}\,. \end{aligned} \tag{17}$$

Eq. (17) is define as the *Gaussian rejection model* for detecting ambiguous category assignments. When implemented as part of a clusterer, μ_j is equal to $\mathbf{w}_j$ and variance σ^2 is required, either from

prior knowledge or on-line estimation. The Euclidean distance is a core component of the prototype matching function commonly used by clusterers when prototypes represent mean patterns. Assessing the ambiguity of category assignments with Eq. (17) is thus relatively straightforward.

4.2.2 Softmax rejection model

If no assumption is made regarding the underlying data distribution, Eq. (12) can be used for direct ambiguity detection. During on-line category learning, the prototype matching function provides the response strength for each category with respect to an input $\mathbf{a}$. The match strength ϕ_j between input $\mathbf{a}$ and prototype $\mathbf{w}_j$ can be interpreted as an estimate of the a posteriori probability $p(j \mid \mathbf{a})$.

To ensure that the ϕ_j values are valid probabilities (*i.e.*, sum up to 1 and range from 0 to 1), the *softmax activation function* [4] from neural network literature is applied. In contrast to the winner-take-all ("hard max") activation function:

$$y_j = \begin{cases} 1; & \text{if } j = \arg\max\{\phi_k : k = 1, 2, ..., N\}, \\ 0; & \text{otherwise} \end{cases} \tag{18}$$

the softmax activation function is expressed as:

$$y_j = \frac{\exp\{\phi_j\}}{\sum_{k=1}^{N} \exp\{\phi_k\}}. \tag{19}$$

Assuming that y_j can be used as an estimate of $p(j \mid \mathbf{a})$, it follows that $r_J(\mathbf{a}) \cong 1 - y_J$, the rejection rule of Eq. (12) can be expressed in a form similar to that of Eq. (13):

$$\begin{aligned} 1 - \frac{\exp\{\phi_J\}}{\sum_{j=1}^{N} \exp\{\phi_j\}} &\geq \gamma \\ \left(\frac{1-\gamma}{\gamma}\right) &\geq \frac{\exp\{\phi_J\}}{\sum_{j=1, j\neq J}^{N} \exp\{\phi_j\}}, \end{aligned} \tag{20}$$

Eq. (20) is define as the *softmax rejection model* for detecting ambiguous category assignments.

5 Simulation results and discussion

5.1 Experimental methodology

For computer simulation, unsupervised competitive learning (UCL) neural networks were used as clusterers with sequential, batch, and reordered processing. In order to compare the performance of these clustering systems, computer simulations were repeated over several independent trials with different presentation orders. Prior to each trial, a fi ed-length data set was normalized using a linear transformation such that the values of every feature ranged from 0 to 1. The normalized patterns were then shuffle into a random presentation order (*i.e.*, random permutations of the patterns, regardless of cluster), and then learned by the clustering system under test. After every trial, clustering quality was measured. This subsection describes the clustering quality measure, the UCL neural networks, and the data sets used.

5.1.1 Rand Adjusted measure of similarity

A partition of n patterns into K groups define a *clustering*. This can be represented as a set $A = \{a_1, a_2, ..., a_n\}$, where $a_h \in \{1, 2, ..., K\}$ is the category label assigned to pattern h. The degree of match between two clusterings, A and B, may be compared by constructing a contingency table (refer to Figure 1). In this table, c_{11} (c_{22}) is the number of pattern pairs that are in a same (different) cluster in both partitions. The value c_{21} is the number of pattern pairs that are placed in a same cluster by A, but in different clusters by B. The value c_{12} reflect the converse situation. The four variables within the contingency table have been used to derive measures of similarity between two clusterings A and B [2] [11]. These measures are known in pattern recognition literature as external criterion indices, and used for evaluating the capacity to recover the true cluster structure.

Table 1: Contingency table used to compare two clusterings.

		Clustering A	
		same	**different**
Clustering B	**same**	c_{11}	c_{12}
	different	c_{21}	c_{22}

Of these measures, the Rand Adjusted [27], define by:

$$S_{RA}(A,B) = \frac{2(c_{11}c_{22} - c_{12}c_{21})}{2c_{11}c_{22} + (c_{11}+c_{22})(c_{12}+c_{21}) + c_{12}^2 + c_{21}^2} \tag{21}$$

has been selected to assess clustering quality in this paper.

Since correct classificatio results are known for the data sets used, their patterns are all accompanied by category labels. These labels are withheld from the network under test, but they provide a reference clustering, R, with which a clustering produced by computer simulation, A, may be compared. In this context, variables c_{11} and c_{22} represent the number of pattern pairs which are properly clustered together and apart, respectively, while c_{12} and c_{21} indicate the improperly clustered pairs. Eq. (21) now represents a *score*, $S_{RA}(A,R)$, that describes the quality of the clustering produced by the network. The Rand Adjusted score ranges from 0 to 1, where 0 denotes maximum dissimilarity (worst), and 1 denotes equivalence (best).

5.1.2 Unsupervised competitive learning neural networks

Self-organizing neural networks use unsupervised learning to discover relationships of interest (correlations, categories, etc.) that may exist in the input data [26] [32]. Unsupervised competitive

learning (UCL) [21] [22] [23] [30] [35] [36] refers to a family of self-organizing neural networks, typically applied to adaptive vector quantization [18] [20] [31] for, *e.g.*, data compression, or to discover clusters in unlabeled data [2] [11] [25].

Structurally, a basic UCL neural network consists of a layer of identical output neurons, each one fully connected to the input neurons (one per input feature) with feedforward weights. A category label is associated with each output neuron, and a prototype — the numerical values of the set of weights connecting an output neuron to all input neurons — represents a category's features. When an input pattern is presented to the network's input neurons, signals are propagated through the feedforward weight connections, and the output neurons compete among themselves for *winner-take-all* activation. The output neuron that is active after the competition is the one whose prototype most closely resembles the input pattern. This neuron is declared the winner, its category is assigned to the input, and then it is allowed to adapt its prototype. From a succession of input patterns, output neurons learn to specialize for regions of the input space [26] [35].

An UCL network applied to high throughput category learning for the two types of environments considered in this paper should deliver a rapid category assignment and prototype adaptation in response to each input pattern. The network's behavior should also be independent of the distribution of input patterns among categories[1]. Finally, the network should learn incrementally, and remain adaptive or plastic to new information which may arise at any time in a non-stationary data stream. Plasticity to new information may cause instability or amnesia in standard UCL networks, since weights can erode over time [23]. Variants such as the Self-Organizing

[1]Independence of behavior to the mixture of patterns assigned to categories precludes the use of a series of UCL extensions, such as the Conscience Method [10] [29] and Frequency Sensitive Competitive Learning [1], that apply a conscience principle [21] [22] to overcome an output neuron under-utilization problem.

Feature Map (SOFM) [30] can achieve stability by monotonic reduction of the learning rate and neighborhood function as learning progresses. In contrast, Adaptive Resonance Theory (ART) networks [5] [21] [22] [23] can develop stable recognition capability on-line. They categorize familiar inputs into previously learned adaptive categories, and create new categories dynamically for inputs different enough from those previously seen. Their weights remain stable, yet plastic in response to new events because they can create new categories, while keeping formerly learned weights intact [22]. A *vigilance* parameter ρ is used to regulate the maximum tolerable difference between any two input patterns to which a category is assigned.

The UCL neural networks selected for computer simulations are (1) standard UCL with Malhalanobis distance, (2) ART2A-E [15], and (3) fuzzy ART [6]. Networks (1) and (2) are well suited for the firs type of environment (described in Sections 4.4.2), where clusters are explicitly modeled as mean vectors. Networks (2) and (3) are appropriate for the second type of environment, where clusters are modeled implicitly. Appendix A highlights the main architectural and algorithmic characteristics of these three neural networks.

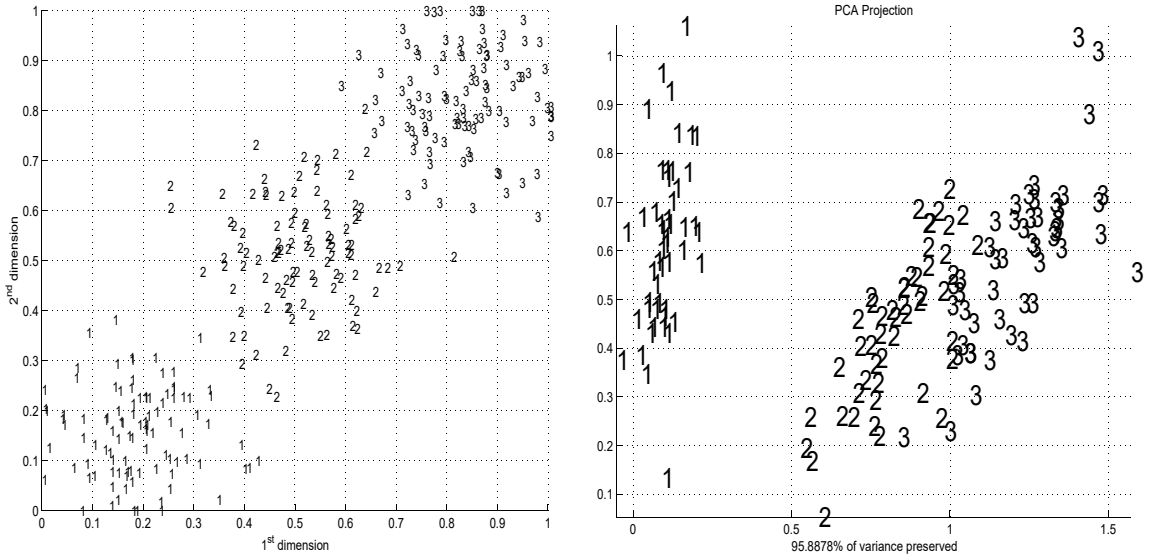
Programs emulating these neural networks were written in the Matlab language for all three data processing schemes. For reordered processing, each network was granted the capacity to detect ambiguity. The Gaussian rejection model (refer to Eq. (17) was used in (1) standard UCL and (2) ART2A-E networks. The softmax rejection model (refer to Eq. (20) was used in (3) ART2A-E and (4) fuzzy ART networks. For standard UCL, $\phi_j = -d_j$, for $j = 1, 2, ..., N$, was substituted into Eq. (20). For ART2A-E or fuzzy ART, $\phi_j = T_j$, for j such that the vigilance test is passed was substituted in Eq. (20).

5.1.3 Data sets

Two data sets were employed: Gaussian and Iris. The Gaussian data set represents the firs type of on-line clustering environment considered in this paper. It is mostly targeted towards clustering systems based on the standard UCL and ART2A-E networks with the Gaussian rejection model. The Iris data set represents the second type of on-line clustering environment. It is targeted towards clustering systems based on the ART2A-E and fuzzy ART networks with the softmax rejection model.

(a) Gaussian data: The Gaussian data set consists of 3 clusters with mean vectors $\mu_1 = (1,1)$, $\mu_2 = (3,3)$ and $\mu_3 = (5,5)$. Each cluster $j = 1,2,3$ is formed by the random generation of 100 patterns from a normal spherical distribution centered at μ_j, with standard deviation $\sigma = 0.6$. A new Gaussian data set was artificiall generated for each simulation trial. Figure 2(a) shows a typical Gaussian data set after normalization.

(b) Iris data: The well-known Iris data set [14] contains parameters from 150 fl wers, each one belonging to one of three species of iris fl wers — satosa, versicolor and virginica. Fifty fl wers belong to each species, and each fl wer is characterized by its petal and sepal length and width (4 features). Figure 2(b) shows a two-dimensional principal component analysis (PCA) projection of the Iris data set after normalization. Data clusters are scattered, and two of them overlap considerably.



(a) Gaussian data.

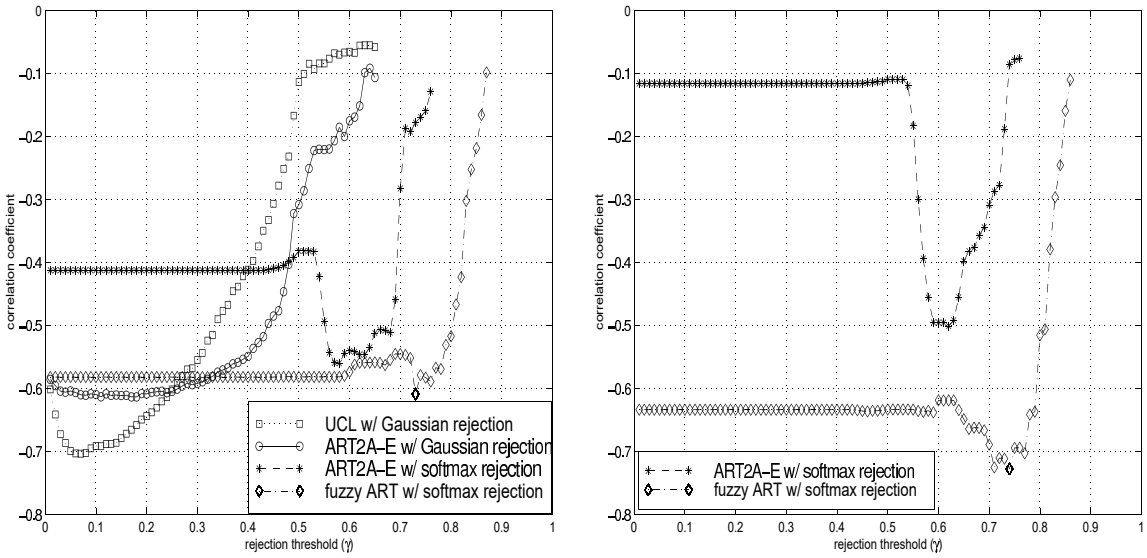
(b) PCA projection of the Iris data.

Figure 2: The two data sets used for computer simulations.

5.2 Correlation of ambiguity with clustering quality

The clustering quality achieved by UCL networks is now examined in relation with the degree of ambiguity observed in their category assignments. Patterns from the Gaussian and Iris data sets were categorized by UCL neural networks using reordered processing. Regular network parameters like the learning rate β were fi ed *a priori*, for each data set, to values that produce a high Rand Adjusted score $S_{RA}(A,R)$ using sequential processing.

For a same randomly-selected presentation order, the rejection threshold γ was varied from trial to trial, in the range from 0 to 1. Patterns **a** leading to ambiguous category assignments were detected, yet processed without delay (the mechanism to postpone patterns was deactivated). After each trial, the score $S_{RA}(A,R)$, and the empirical rejection rate $\hat{p}_r$ were computed and stored. The empirical rejection rate $\hat{p}_r$ is the ratio of the number of rejected patterns to the total number of



(a) Gaussian data set.

(b) Iris data set.

Figure 3: Correlation between the Rand Adjusted score, $S_{RA}(R, A)$, and the empirical rejection rate, $\hat{p}_r$, for three clusterers, two rejection criteria and two data sets. Notice that Iris data simulations do not apply when UCL networks are augmented with Gaussian rejection, since category spread information (σ) is unavailable.

patterns (the size of the data set). As mentioned in Section 4.3, the number of rejections generally decreases as the threshold γ is increased. In the context of on-line category learning, the number of rejections varies according to the order in which data are presented. This number also varies with different random generations of Gaussian data. The linear correlation between scores $S_{RA}(A, R)$ and respective rates $\hat{p}_r$ was computed from a series of 1000 independent presentation orders, with γ ranging from 0 to 1.

The linear correlation coefficient obtained with the Gaussian and Iris data sets are shown in Figure 3(a) and (b), respectively. Table 2 documents the peak negative correlation coefficien

values in Figure 3 for future reference. Results reveal that, for all the clustering systems, and for both data sets, there is a significan negative correlation between $\hat{p}_r$ and $S_{RA}(A,R)$. From one trial to the next, and for a given threshold value γ, an increase in the number of ambiguous category assignments corresponds to a decline of the clustering score.

The four clustering systems display strong negative correlations, albeit in different ranges of values for the threshold γ. In simulations involving the Gaussian data, the Gaussian rejection define relatively thin ambiguity regions owing to the explicit use of category spread information ($\sigma^2 = 0.36$ in Eq. (17)). The conditional error function $r_J(\mathbf{a})$ associated with the Gaussian rejection criterion Eq. (11), which define the ambiguity region, declines relatively quickly at the decision boundary of category J. Consequently, the two UCL networks that use Gaussian rejection achieve strong negative correlations at low threshold values ($\gamma < 0.4$).

In contrast, softmax rejection does not use category variance information, and thus define relatively wide ambiguity regions. The conditional error function $r_J(\mathbf{a})$ associated with the softmax rejection criterion (Eq. (20)) declines relatively slowly at the decision boundary of category J. Furthermore, the ART networks have a tendency to create more than three categories to represent the data clusters in Gaussian and Iris sets. Consequently, ART2A-E achieves strong negative correlations at high threshold values ($\gamma \in [0.55 - 0.70]$) for both data sets. It is interesting to note that, despite making abstraction of σ information, softmax rejection can provide a negative correlation almost equivalent in strength to that obtained with Gaussian rejection.

Table 2: Peak negative correlation coefficien values associated with Figure 3. The rejection threshold value γ, empirical rejection rate $\hat{p}_r$ (with standard error in parentheses) and parameter settings that correspond to these peak correlations are also shown.

Neural Networks	Correlation Coefficien	
	Gaussian data	**Iris data**
Standard UCL (Malhalobis dist.) with Gaussian rejection	**-0.7034** $\gamma = 0.07 \rightarrow \hat{p}_r = 8.0\%$ (0.1%) ($\beta = 0.50$, 3 neurons, $\sigma = 0.60$)	**N/A**
ART2A-E with Gaussian rejection	**-0.6133** $\gamma = 0.18 \rightarrow \hat{p}_r = 4.3\%$ (0.1%) ($\beta = 0.50$, $\rho = 0.70$, $\sigma = 0.60$)	**N/A**
ART2A-E with softmax rejection	**-0.5596** $\gamma = 0.58 \rightarrow \hat{p}_r = 8.0\%$ (0.4%) ($\beta = 0.50$, $\rho = 0.70$)	**-0.5007** $\gamma = 0.62 \rightarrow \hat{p}_r = 15.3\%$ (0.1%) ($\beta = 0.45$, $\rho = 0.70$)
Fuzzy ART with softmax rejection	**-0.6087** $\gamma = 0.73 \rightarrow \hat{p}_r = 35.7\%$ (0.6%) ($\beta = 0.90$, $\rho = 0.60$, $\alpha = 1$)	**-0.7275** $\gamma = 0.74 \rightarrow \hat{p}_r = 24.7\%$ (0.1%) ($\beta = 0.85$, $\rho = 0.60$, $\alpha = 1$)

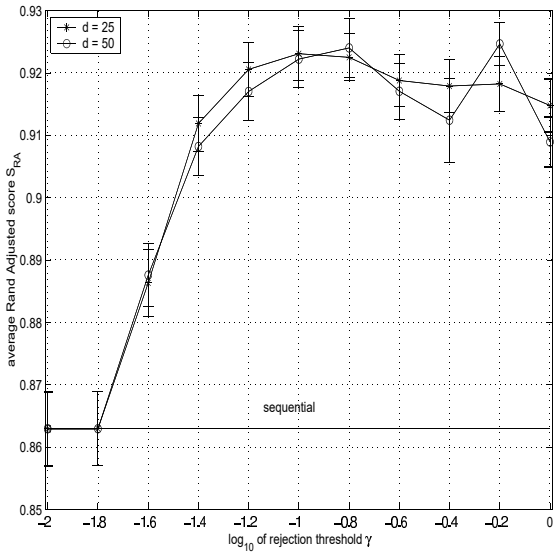
Fuzzy ART using the softmax rejection model displays strong negative correlation across a wide range of threshold values. Its ability to assess ambiguity appears less sensitive to input data structure, as it provides consistently strong negative correlations for both data sets. This can be explained by the fact that fuzzy ART is the most susceptible of the three UCL networks to ambiguity. Indeed, fuzzy ART is unable to forget, and hence recover from category miss-assignments.

5.3 Clustering quality obtained using reordered processing

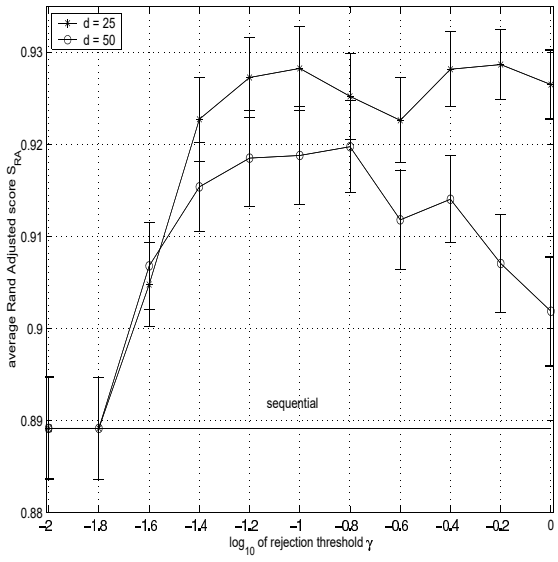
Having shown that the Gaussian and softmax rejection models detect ambiguous category assignments, and that these assignments lead to poor clustering performance, the effect of delaying such assignments is now examined.

For a same randomly-selected presentation order, the rejection threshold γ was varied from 0 to 1, over successive trials. During each trial, patterns from either the Gaussian or Iris data set were presented to a clustering system used with reordered processing. As in Subsection 4.5.2, the network parameters were fi ed *a priori* to provide high Rand Adjusted scores $S_{RA}(R,A)$ using sequential processing. Rejected patterns were delayed by d patterns. Two different delay values, $d = 25$ and 50, were tried. After each trial, the score, $S_{RA}(A,R)$ and empirical rejection rate, $\hat{p}_r$, were computed and stored. Average $S_{RA}(A,R)$ and $\hat{p}_r$ values were obtained from a series of 20 independent presentation orders, with γ ranging from 0 to 1.

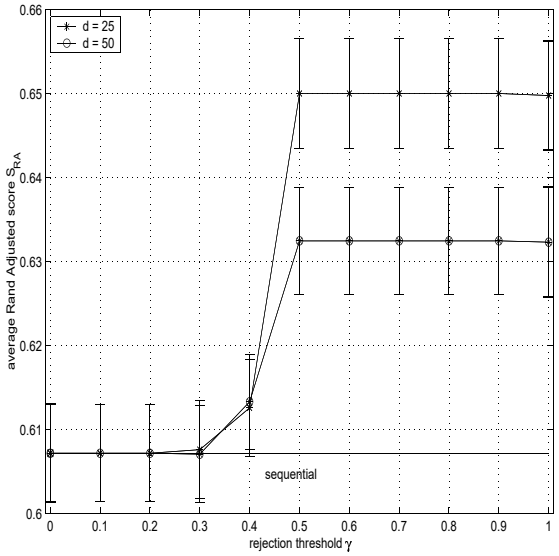
Figure 4 shows the average $S_{RA}(A,R)$ as a function of γ for four different cases — standard UCL and ART2A-E on Gaussian data, and ART2A-E and fuzzy ART on Iris data. Figure 5 shows the same average $S_{RA}(A,R)$ as a function of the average $\hat{p}_r$. In all cases, reordered processing increases



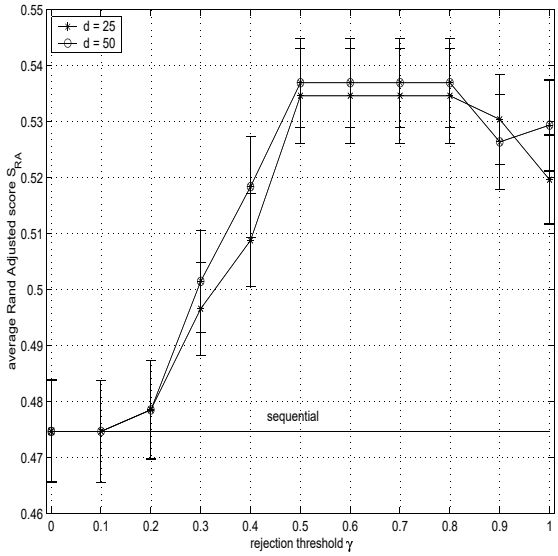
(a) standard UCL on Gaussian data.



(b) ART2A-E on Gaussian data.

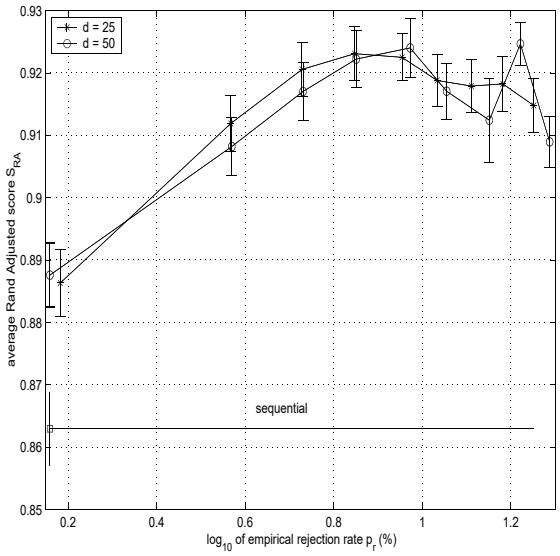


(c) ART2A-E on Iris data.

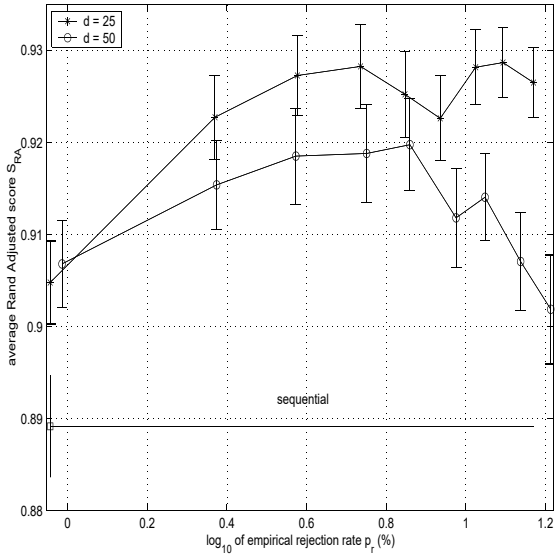


(d) fuzzy ART on Iris data.

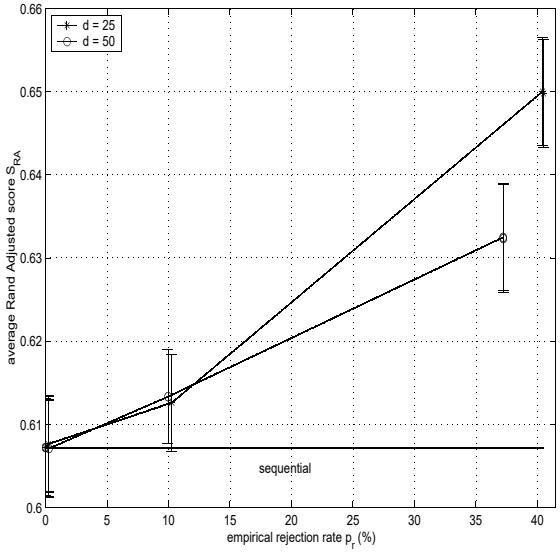
Figure 4: Average Rand Adjusted scores $S_{RA}(A,R)$ versus rejection threshold γ for simulations with Gaussian and Iris data. (Error bars are standard error.)



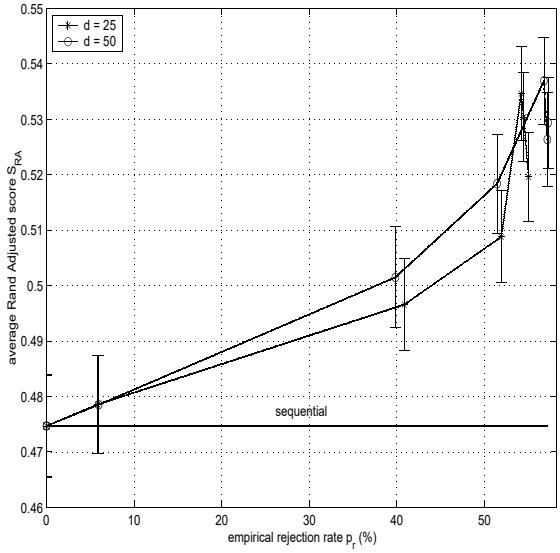
(a) standard UCL on Gaussian data.



(b) ART2A-E on Gaussian data.



(c) ART2A-E on Iris data.



(d) fuzzy ART on Iris data.

Figure 5: Average Rand Adjusted scores $S_{RA}(A,R)$ versus empirical rejection rate $\hat{p}_r$ for simulations with Gaussian and Iris data. (Error bars are standard error.) The standard error for values of $\hat{p}_r$ always range from 0% to 2%.

the clustering score significantly over sequential processing alone. For example, if pattems from the Iris data are presented to the clustering system with fuzzy ART and d = 50, then reordered sequential processing improves the score $S_{RA}(A,R)$ by about 13%. Clustering improvements are achieved over a wide range of y values. In the example given above (fuzzy ART on the Iris data), the maximum score of $S_{RA}(A,R) \cong 0.538$ is obtained for $\gamma \in [0.5, 0.8]$, corresponding to $\hat{p}_r \cong 55\%$ in Figure 5. For $0 \leq$ y < 0.5, the performance tends towards that of sequential processing since $\hat{p}_r \to 0\%$. With y > 0.8, the performance also tends to decline because of the large number of rejections. Notice that improvements to S_{RA} *(A, R)* occur when there is a strong negative cor-relation between $\hat{p}_r$ and S_{RA} *(A, R)*.

Of the four cases, reordered processing appears most beneficial to a clustering system with fuzzy ART, where irreversible leaming has the strongest impact. To ensure stability, fuzzy ART weights are adjusted such that each category hyperrectangle can only grow or remain the same. In contrast, standard UCL and ART2A-E represent categories as mean vectors, and obtain their best performance at slower leaming rates, which allow to "forget" some category miss-assignments due to ambiguity.

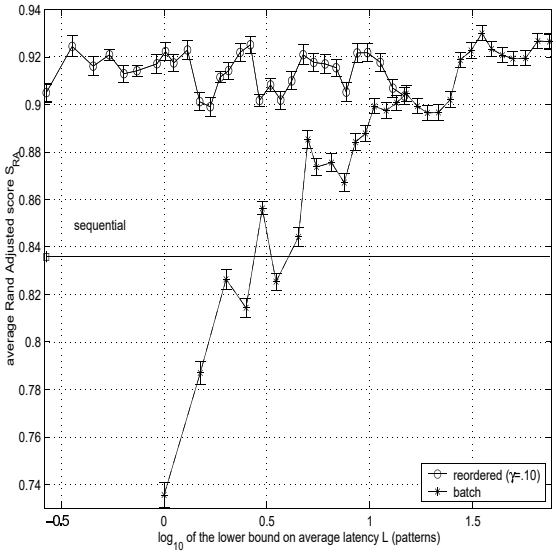
5.4 Clustering quality and latency

For fast on-line clustering applications, the quality of results must be balanced with the response time needed to produce the results. The performance of different clustering systems is now compared in terms of clustering quality and latency.

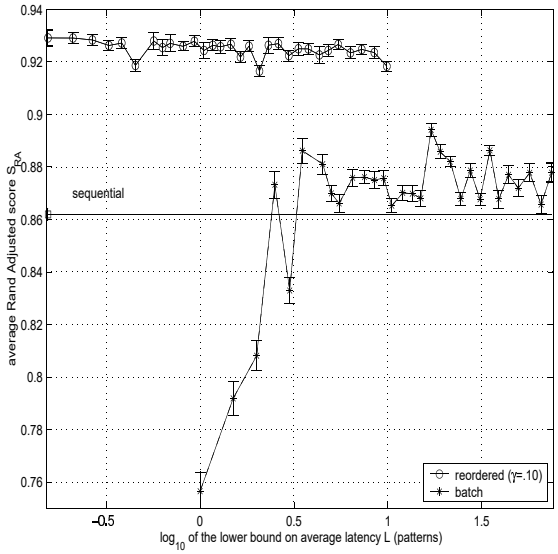
For a same randomly-selected presentation order, the queue lengths k and d were varied between 5 pattems and half the number of pattems in the data set, successive trials. This range

guarantees at least two batches of k patterns for batch processing, and limits the amount of patterns being postponed at any given time for reordered processing. During each trial, patterns from either the Gaussian or Iris data set were presented to the batch and reordered processing architectures. With the batch processing architecture, the UCL network was left to converge for successive batches of k patterns, until prototype weights were identical for two consecutive epochs. Regular network parameters were fi ed *a priori* to provide high Rand Adjusted scores $S_{RA}(R,A)$ with batch processing when k is one half the data set size. With the reordered processing architecture, a different rejection threshold value γ was selected for each UCL network. Regular network parameters were fi ed *a priori* to provide high scores $S_{RA}(R,A)$ with sequential processing. Rejected input patterns were delayed for d patterns. After each trial, the score $S_{RA}(R,A)$, the lower bound on the average latency $\overline{L}$ (Eq. (3)), and the average latency $\overline{L}$ of the considered architectures (Eq. (7) for the batch case) were computed. The empirical rejection rate, $\hat{p}_r$, provides a fi ed estimate of the variable rejection rate $p_r(\mathbf{a})$, and was substituted into Eq. (3). Simulations were repeated 20 times for every k and d value in order to yield representative results.

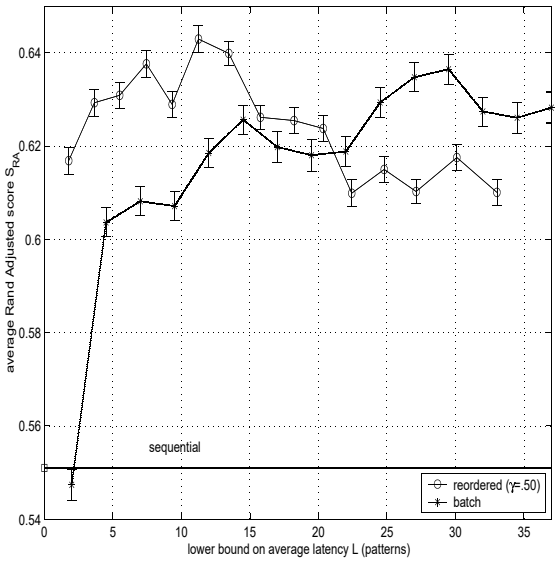
The average scores $S_{RA}(A,R)$ as a function of the lower bounds on average latency $\overline{L}$ are shown in Figure 6 for four different cases — standard UCL and ART2A-E on Gaussian data, and ART2A-E and fuzzy ART on Iris data. Both batch and reordered processing achieve significantl higher scores than sequential processing. However, reordered processing achieves the improvement at a much lower average latency than batch processing. For example, with fuzzy ART on the Iris data, reordered processing with $\gamma = 0.65$ yields a score of $S_{RA}(A,R) \cong 0.534$ for an average latency of $\overline{L} \cong 6.4$ patterns. This corresponds to $\hat{p}_r \cong 42.5\%$ and $d = 15$. By comparison, batch processing yields $S_{RA}(A,R) \cong 0.551$ for an average latency of $\overline{L} \cong 32$ patterns. This corresponds to $k = 65$.



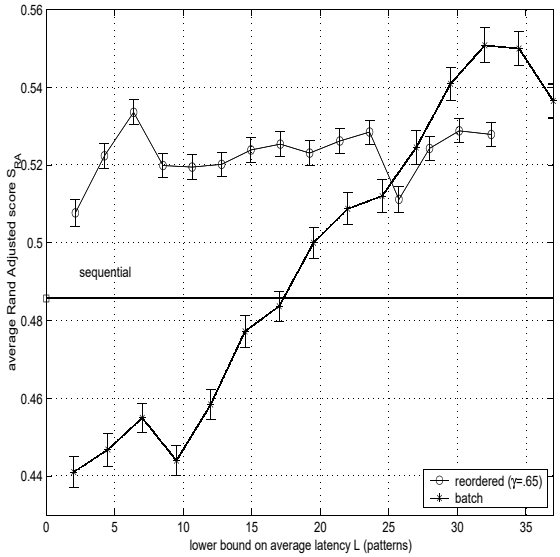
(a) standard UCL on Gaussian data.



(b) ART2A-E on Gaussian data.

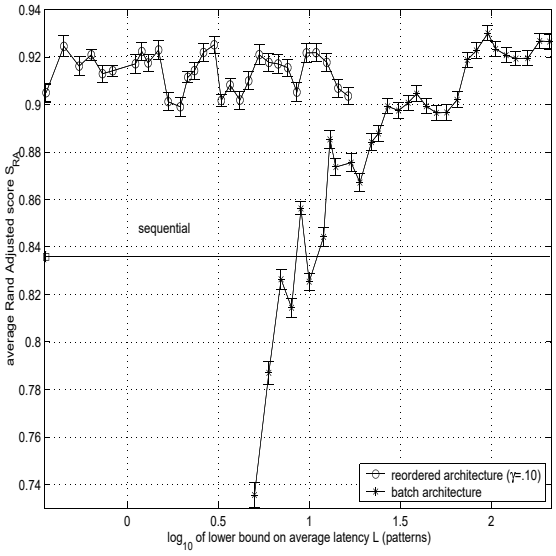


(c) ART2A-E on Iris data.

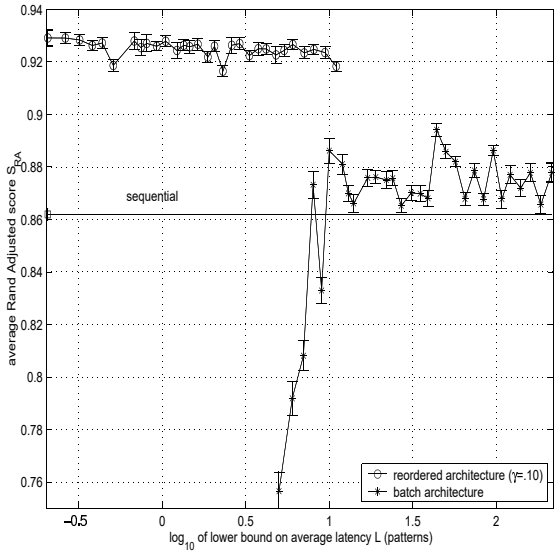


(d) fuzzy ART on Iris data.

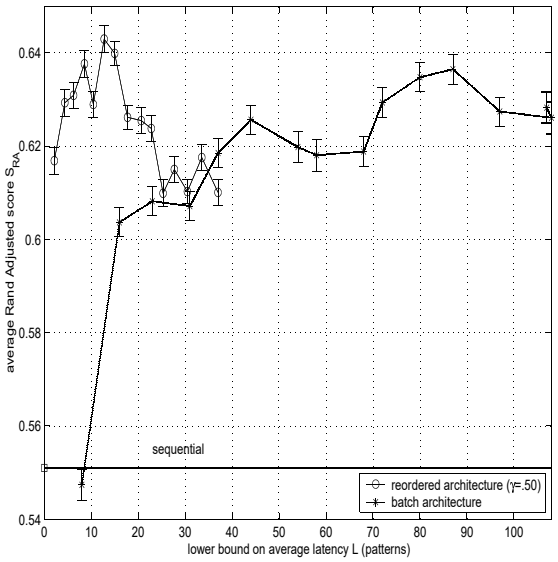
Figure 6: Average Rand Adjusted scores $S_{RA}(A,R)$ versus lower bound on average latency $\overline{L}$ for simulations with Gaussian and Iris data. (Error bars are standard error.)



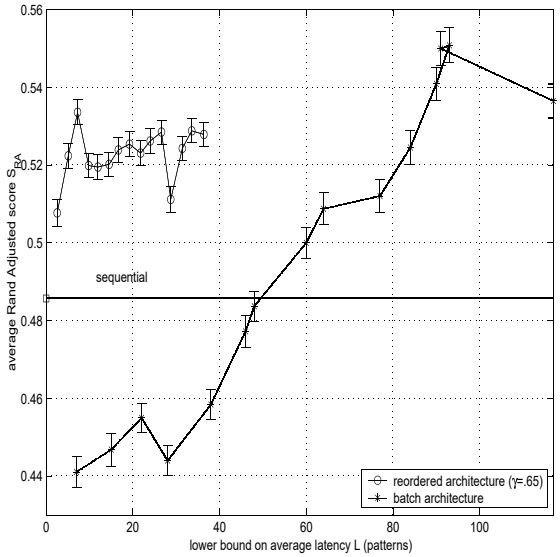
(a) standard UCL on Gaussian data.



(b) ART2A-E on Gaussian data.



(c) ART2A-E on Iris data.



(d) fuzzy ART on Iris data.

Figure 7: Average Rand Adjusted scores $S_{RA}(A,R)$ versus average latency $\overline{L}$ of the batch and reordered architectures for simulations with Gaussian and Iris data. (Error bars are standard error.)

Results in Figure 6 reveal that the performance obtained with batch processing increases with k until a peak value. Peak performance with reordered processing sometimes exceed that of batch processing. It often occurs for relatively small values of d, indicating that it is suitable for high speed applications. Beyond this peak, increasing d does not necessarily deliver a greater score $S_{RA}(A,R)$. This is due to our emulation of an infinit data stream with a fi ed-sized data set. If an input **a** to be processed is among the last d patterns of a data set, and it is rejected by the clusterer, then it is postponed to the end of the data set (following any other previously-rejected patterns), and may be postponed for less than d patterns. As **a** is located closer to the end of the data set, and as γ and d grow, it is more likely for **a** to be postponed by less than d patterns. Clustering quality accordingly converges towards that of sequential processing.

The average scores $S_{RA}(A,R)$ as a function of the average latency $\overline{L}$ for the batch and reordered architectures are shown in Figure 7. With the batch architecture, there is a substantial difference between the curves obtained with $\overline{L}$ and those obtained with the lower bound on $\overline{L}$. Indeed, using the batch processing architecture requires between 2 and 5 epochs for convergence on each batch of patterns. Assuming that the techniques used to assess ambiguity have a relatively low computational complexity, the average latency of the reordered architecture compares favorably to that of the batch architecture.

6 Conclusions

Both clustering quality and response time are critical in many on-line category learning applications. A clustering system targeted for high throughput applications would traditionally consist of a clusterer that processes input patterns sequentially. Despite the fast response time obtained when

using this data processing scheme, the quality and consistency of the results remain an issue. By contrast, batch processing yields higher clustering quality, but incurs longer delays.

In this paper, an alternative called reordered processing has been presented. It consists in postponing for a fi ed time the learning of patterns for which category assignments are ambiguous. Ambiguity occurs when the clusterer lacks the information needed to take a fir decision in favor of a single category. The reject option from statistical pattern recognition literature has been extended for the detection of ambiguous category assignments. Reordered processing alters the original pattern presentation order in an attempt to avoid ambiguous decisions. In addition, it provides control of the maximum response time of a clustering system. Latency of on-line category learning has been define and used to compare sequential, batch or reordered processing. In order to reveal some of the practical considerations of implementing batch and reordered processing, the latency of two high level architectures have also been described.

Computer simulations were performed by presenting patterns from two data sets to several neural networks with sequential, batch and reordered processing. Results of these simulations indicate that (1) the number of category assignments detected as ambiguous is correlated with poor clustering quality; (2) reordered processing improves clustering quality over sequential processing alone and sometimes over batch processing; and (3) it offers an interesting alternative to batch processing in terms of trading off clustering quality for response time.

Acknowledgements

This research was supported in part by the Fonds pour la Formation de Chercheurs et l'Aide à la Recherche du Québec, and the Natural Sciences and Engineering Research Council of Canada.

References

[1] Ahalt, S. C., Krishnamurthy, A. K., Chen, P., and Melton, D. E., "Competitive Learning Algorithms for Vector Quantization," *Neural Networks*, **3**, 277-290, 1990.

[2] Anderberg, M. R., *Cluster Analysis for Applications*, London, UK: Academic Press, 1973.

[3] Bishop, C., *Neural Networks for Pattern Recognition*, Oxford, UK: Clarendon Press, 1995.

[4] Bridle, J. S., "Probabilistic Interpretation of Feedforward Classificatio Network Outputs, with Relationships to Statistical Pattern Recognition," in *Neurocomputing – Algorithms, Architectures and Applications*, eds., Fogelman-Soulié, F., Hérault, J., NATO ASI Series F68, Berlin: Springer-Verlag, 227-236, 1989.

[5] Carpenter, G. A. and Grossberg, S., "A Massively Parallel Architecture for a Self-Organizing Neural Pattern Recognition Machine," *Computer, Vision, Graphics and Image Processing*, **37**, 54-115, 1987.

[6] Carpenter, G. A., Grossberg, S., and Rosen, D. B., "Fuzzy ART: Fast Stable Learning and Categorization of Analog Patterns by an Adaptive Resonance System," *Neural Networks*, **4:6**, 759-771, 1991.

[7] Chow, C. K., "An Optimum Character Recognition System Using Decision Functions," *IRE Trans. Electronic Computers*, **6**, 247-254, 1957.

[8] Chow, C. K., "On Optimum Recognition Error and Reject Tradeoff," *IEEE Trans. Information Theory*, **16**, 41-46, 1970.

[9] Davies, C. L. and Hollands, P., "Automatic Processing for ESM," *Proc. IEE*, **129:3 (F)**, 164-171, June 1982.

[10] DeSieno, D., "Adding a Conscience to Competitive Learning," *Proc. IEEE Int'l Conf. Neural Networks*, 1117-1124, San Diego, June 1988.

[11] Dubes, R. C. and Jain, A. K., *Algorithms for Clustering Data*, Englewood Cliffs, NJ:Prentice Hall, 1988.

[12] Dubuisson, B. and Masson, M., "A Statistical Decison Rule with Incomplete Knowledge About Classes," *Pattern Recognition*, **26:1**, 155-165, 1993.

[13] Duda, R. O., and Hart, P. E., *Pattern Classificatio and Scene Analysis*, John Weiley & Sons, 1973.

[14] Fisher, R. A., "The Use of Multiple Measurements in Taxonomic Problems," *Annals of Eugenics*, **7**, 1936.

[15] Frank, T., Kraiss, K.-F., and Kuhlan, T., "Comparative Analysis of Fuzzy ART and ART-2A Network Clustering Performance," *IEEE Trans. Neural Networks*, **9:3**, 544-559, 1998.

[16] Fukunaga, K. and Kessell, D. L., "Application of Optimal Error-Reject Functions," *IEEE Trans. Information Theory*, 814-817, November 1972.

[17] K. Fukunaga, *Introduction to Statistical Pattern Recognition*, Academic Press, 2nd ed., 1990.

[18] Gersho, A., "On the Structure of Vector Quantizers," *IEEE Trans. Information Theory*, **28:2**, 157-166, 1982.

[19] Granger, E., Savaria, Y., Lavoie P., and Cantin, M.-A., "A Comparison of Self-Organizing Neural Networks for Fast Clustering of Radar Pulses," *Signal Processing*, **64:3**, 249-269, 1998.

[20] Gray, R. M., "Vector Quantization," *IEEE ASSP Magazine*, **1:2**, 4-29, 1984.

[21] Grossberg, S., "Adaptive Pattern Classificatio and Universal Recoding: I. Parallel Development and Coding of Neural Detectors," *Biological Cybernetics*, **23**, 121-134, 1976.

[22] Grossberg, S., "Adaptive Pattern Classificatio and Universal Recoding: II. Feedback, Oscillation, Olfaction, and Illusions," *Biological Cybernetics*, **23**, 187-207, 1976.

[23] Grossberg, S., "Competitive Learning: From Interactive Activation to Adaptive Resonance," *Cognitive Science*, **11**, 23-63, 1987.

[24] Ha, T. M., "The Optimal Class-Selective Rejection Rule," *IEEE Trans. Pattern Analysis and Machine Intelligence*, **19:6**, 608-615, 1997.

[25] Hartigan, J. A., *Clustering Algorithms*, New York, NY: Wiley, 1975.

[26] Haykin, S., *Neural Networks: A Comprehensive Foundation*, Macmillan College Press, 1994.

[27] Hubert, L. and Arabie, P., "Comparing Partitions," *Journal of Classificatio* , **2**, 193-218, 1985.

[28] Hellman, M. E., "The Nearest Neighbor Classificatio Rule with Reject Option," *IEEE Trans. Systems Science and Cybernetics*, **6:3**, 179-185, 1970.

[29] Hecht-Nielsen, R., "Applications of Counterpropagation Networks," *Neural Networks*, **1**, 131-141, 1988.

[30] Kohonen, T., *Self-Organization and Associative Memory*, Springer-Verlag, 3rd ed., 1989.

[31] Linde, Y., Buzo, A., and Gray, R. M., "An Algorithm for Vector Quantizer Design," *IEEE Trans. Communications*, **28:1**, 84-95, 1980.

[32] Lin, C.-T. and George Lee, C. S., *Neural Fuzzy Systems: A Neuro-fuzzy Synergism to Intelligent Systems*, Prentice-Hall, 1995.

[33] Lloyd, S. P., "Least-square Quantization in PCM," *IEEE Trans. Information Theory*, **28:2**, 129-137, 1982.

[34] MacQueen, J., "Some Methods for Classificatio Analysis of Multivariate Observations," *Proc. 5th Berkley Symposium on Mathematical Statistics and Probability*, 281-297, 1967.

[35] Rumelhart, D. E. and Zipser, D., "Feature Discovery by Competitive Learning," *Cognitive Science*, **9**, 75-112, 1985.

[36] Von der Malsberg, C., "Self-Organizating of Orientation Sensitive Cells of the Striate Cortex," *Kybernetik*, **14**, 85-100, 1973.

A Summary of UCL neural networks

A.1 Neural network structure

A simplifie UCL neural network (refer to Figure 8) consists of two layers of neurons that are fully connected: an M neuron input layer, F_1, and an N neuron output or *competitive* layer, F_2. Each F_1 neuron i is associated with an input feature, whereas every F_2 neuron j represents a category in the input space. A set of adaptive weights $\mathbf{W} = \{w_{ij} : i = 1,2,...,M; j = 1,2,...,N\}$ is associated with the F_1-to-F_2 layer connections. Weight value w_{ij} is represented as a real in the interval [0,1]. Finally, as shown in the shaded parts of Figure 8, ART networks also require circuitry to perform vigilance testing (refer to Section 4.A.2).

For each neuron j of F_2, the *prototype* $\mathbf{w}_j = (w_{1j}, w_{2j}, ..., w_{Mj})$ define the characteristics of category j. With standard UCL and ART2A-E networks, each prototype represents a center point in the input scene (computed from patterns assigned to that category). In this respect, fuzzy ART differs significantl from the other two, since prototype vectors represent fuzzy set hyperrectangles.

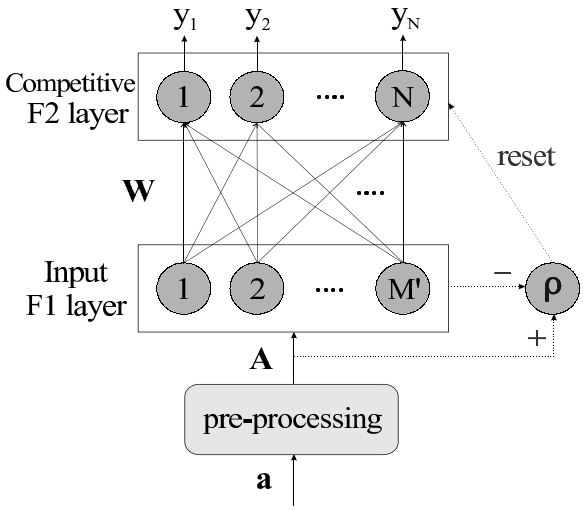


Figure 8: Structure of an UCL neural network.

A.2 Algorithmic description

Neural network processing can be define in terms of algorithmic steps. The following algorithm describes the operation of the UCL networks in general terms. Table 3 provides the specifi equations that extend this general description to the standard UCL with Malhalobis distance, ART2A-E and fuzzy ART networks.

1. Weights and parameters initialization: An F_2 neuron j is labeled as committed after being assigned to an m-dimensional input pattern $\mathbf{a} = (a_1, a_2, ..., a_m)$. With standard UCL networks, a fi ed number N of F_2 neurons is committed in advance. Respective prototypes $\mathbf{w}_j$ are preset to the value of N randomly-selected input vectors $\mathbf{a}$. With ART networks, all F_2 neurons are initially uncommitted, but the number of committed F_2 neurons grows progressively. The learning rate β, vigilance ρ and choice α parameters are define at this step.

2. Input preprocessing: When a new input vector $\mathbf{a}$ is presented to the network, it may undergo a preliminary coding, such as normalization. With fuzzy ART, a transformation called *complement coding* doubles the number of components in the input pattern ($M = 2m$), which becomes $\mathbf{A} = (\mathbf{a}; \mathbf{a}^c) = (a_1, a_2, ..., a_m; a_1^c, a_2^c, ..., a_m^c)$, where $a_i^c = (1 - a_i)$. With complement coding, fuzzy ART represents category j as a hyperrectangle that just encloses the training set patterns $\mathbf{a}$ to which it has been assigned. That is, a M-dimensional prototype $\mathbf{w}_j = (w_{1j}, w_{2j}, ..., w_{Mj})$ records the largest and smallest component values of training set patterns $\mathbf{a}$ placed in the j^{th} category.

3. Category assignment: Input pattern $\mathbf{A}$ activates layer F_1 and is propagated through weighted connections $\mathbf{W}$ to layer F_2. Each F_2 neuron j activates according to some proximity measure or

choice function. The F_2 layer produces a binary, winner-take-all pattern of activity $\mathbf{y} = (y_1, y_2, ..., y_N)$, where only the neuron $j = J$ with the greatest activation value remains active ($y_J = 1$). If the greatest activation value is obtained by two or more neurons, the one with the smallest index j wins. In addition, the winning neuron J is subjected to a *vigilance test* in ART networks. If this test is passed, then neuron J remains active and resonance occurs. Otherwise, the network inhibits the active F_2 neuron and searches for another neuron J that passes the vigilance test. If such a neuron does not exist, an uncommitted F_2 neuron becomes active and undergoes prototype update.

4. Prototype learning: The prototype $\mathbf{w}_J$ of the winning neuron J is adjusted to account for input $\mathbf{A}$. The algorithm can be set to slow learning, with $0 < \beta < 1$, or to fast learning, with $\beta = 1$. Once prototype update is performed, the network is ready to process another input pattern (at Step 2).

Table 3: Distinctive equations used by standard UCL, ART2A-E and fuzzy ART networks. Operator $|\cdot|$ is the L_1 norm, $|\mathbf{w}_j| \equiv \sum_{i=1}^{M} |w_{ij}|$, whereas $||\cdot||$ is the Euclidean distance or L_2 norm, $||\mathbf{A}-\mathbf{w}_j|| = \sqrt{\sum_{i=1}^{M}(A_i - w_{ji})^2}$. With fuzzy ART, $\wedge$ is the fuzzy AND operator, $(\mathbf{A}\wedge\mathbf{w}_j)_i \equiv \min(A_i, w_{ij})$.

Algorithmic Step	**Neural Network**		
	UCL with Malhalanobis distance	**ART2A-E**	**fuzzy ART**
1. Initialization:	$\beta, \sigma \in [0,1]$	$\beta, \rho \in [0,1]$	$\alpha > 0$, $\beta, \rho \in [0,1]$
2. Input preprocessing:	$\mathbf{A} = \mathbf{a}$ $(M = m)$		$\mathbf{A} = (\mathbf{a}; \mathbf{a}^c)$ $(M = 2m)$
3. Category assignment:			
$\rightarrow$ choice function $(\forall j)$	$d_j = \dfrac{\|\|\mathbf{A}-\mathbf{w}_j\|\|^2}{\sigma^2}$	$T_j = 1 - \dfrac{\|\|\mathbf{A}-\mathbf{w}_j\|\|}{\sqrt{M}}$	$T_j = \dfrac{\|\mathbf{A}\wedge\mathbf{w}_j\|}{\alpha + \|\mathbf{w}_j\|}$
$\rightarrow$ winner-take-all choice	$J = \arg\min\{d_j : j = 1,2,...,N\}$	$J = \arg\max\{T_j : j = 1,2,...,N\}$	
$\rightarrow$ vigilance test	N/A	$T_J \geq \rho$	$\|\mathbf{A}\wedge\mathbf{w}_J\| \geq \rho M$
4. Prototype learning:	$\mathbf{w}'_J = \beta\mathbf{A} + (1-\beta)\mathbf{w}_J$ $= \mathbf{w}_J + \beta(\mathbf{A}-\mathbf{w}_J)$		$\mathbf{w}'_J = \beta(\mathbf{A}\wedge\mathbf{w}_J) + (1-\beta)\mathbf{w}_J$ $= \mathbf{w}_J + \beta[(\mathbf{A}\wedge\mathbf{w}_J) - \mathbf{w}_J]$